Subject: JEDI User's Guide, D40 (v3)
Author: Zhang Fan Xing, Haley Nguyen, Cecilia Cheng
Date: July 5, 2011

1. Background

JEDI stands for Java EDR (Experimental Data Record) Display Interface. It is an essential EDR quick-view tool that has made important contributions to many past and current NASA/JPL space missions, such as Cassini, MER and Phoenix. However, being a monolithic java-based web application with tightly-coupled client and server components, it showed a sign of technical aging with increasing difficulty for mission adaptation. To better support future space missions, a task calling for a new design and re-implementation was defined and further funded by the MGSS office.

2. Architecture

Listed below are the main features of JEDI:

1. There is a complete separation of presentation from data and logic, which results in independent client and server sides, communicating only through clearly-defined APIs. As such, either side can evolve in its own path, thus is better positioned for easier and much more flexible updates with future web.

2. Some data models in JEDI contain rich science domain knowledge. They have been preserved and improved. Web functionalities are created from scratch leveraging on contemporary web technology.

3. The server side provides individual services covering login/logout, create/display/delete show, etc., all through well-defined APIs. These services can be consumed by the JEDI default client and any new future in-browser or desktop client.

4. The default JEDI client is an AJAX web client application that mimics JEDI client. It works in all major web browsers out of the box without any library dependency.

3. Server

The server is configured as a web application with a client interface. All images and files must be accessible from the web server installed. Authentication and authorization can be configured. Authentication is done via Kerberos verification. Authorization is done via LDAP verification.

The client interfaces with the server via a web interface to create the view needed by a project.

The JEDI server provides individual RESTful services that client interfaces with to create a complete application for the end user.

3.2 Services

Listed below are service endpoints that are supported in current implementation:

- http://hostname:port/jujube/login
- http://hostname:port/jujube/list
- http://hostname:port/jujube/create
- http://hostname:port/jujube/modify
- http://hostname:port/jujube/delete
- http://hostname:port/jujube/logout
- http://hostname:port/jujube/streamAppender
- http://hostname:port/jujube/play
- http://hostname:port/jujube/show

Most of them are access-controlled by auth credentials.

3.2 Web pages

Pages that are parts of the webapp:

- http://hostname:port/jujube/login.html
- http://hostname:port/jujube/list.html
- http://hostname:port/jujube/play.html

4. Client

The client uses a web interface to create a slideshow of images to be displayed. Each image file (FITS, ISIS, VICAR formats) is displayed together with meta information that is customized with a personality file. The image source can be from a list, a directory, or a stream.

4.1. Authentication and Authorization

Only an authenticated and authorized (A&A) user is permitted. A user must provide valid username and password to login. Authentication is done via Kerberos verification. Authorization is done via LDAP verification. If A&A is not activated, the user still has to provide a non-empty username and password. A&A can be turned on or off by modifying the tomcat5.conf file and passing the following parameter to tomcat when it starts:

-Djujube.auth=[true I false]

The LDAP server can be specified in the tomcat5.conf file as well:

```
-Djujube.ldap.url=ldap://miplauth-
dev.jpl.nasa.gov:389/dc=mipl,dc=jpl,dc=nasa,dc=gov
```

To login, use [hostname:port]/jujube/login.html. The default port is 8080.

Project	Environment	URL
Cassini	Development	http://tpsweb-dev/jujube/login.html
Cassini	Test	http://tpsweb-test/jujube/login.html
Cassini	Operations	https://tpsweb/jujube/login.html

Please contact your project representative to set up authentication and

authorization credentials. The installation procedures is documented in Appendix C.

4.2. Create Show

After logging in, the user is presented with a form to create a show.

A show is created using either of two types of data sources:

- a file directory accessible to the server machine that the JEDI application is hosted.
    - Try: /proj/msl/dev/workspace/opgs/matis/ATLO_9/edrgenpipe/output/
    - *VIC
    - /home/hbm/work/MSL/JEDI/MSL_pers1.txt (can be a personality pointer file)
- an in-memory stream data source that can be populated and updated by a tlmproc process via URL push. Requires running the script called JediNewEdr, located in the OPGS tools location (for now, needs to be delivered).
- a list of files

A user can create an arbitrary number of shows, private to the user. Created shows are listed in a table at the top of the page. Each row corresponds to a show that can be played and deleted. If a Show Name is not provided, a show is tagged (and, currently, thus-named) by a combination of user name and creation time. In the current implementation, shows persist only in web server memory and will disappear after a server reboot. It is planned to make them survivable over server reboots.

4.2.1. Select Role

The list of roles for the user is retrieved from the LDAP database. If authentication has been turned off, then there is only 1 role: PROJ. Select the role for your show.

4.2.2. Select Data Source

A show is created using either of two types of data sources:

- A file listing the names of the image files and an optional personality filename, or a directory accessible to the server machine that the JEDI application is hosted.
- An in-memory stream data source that can be populated and updated by the EDR generation pipeline or process via URL push, using a script called JediNewEdr. A stream stays alive (selectable) for 24 hours (default). This parameter can be overridden in the tomcat5.conf:
  `-Djujube.stream.idle=8`

4.2.3. Select Personality

The user has two choices for specifying personality:

- Use the default personality that has been defined in the LDAP database for the role. This is only valid when authorization has been enabled.
- Use a specific personality or a personality pointer file.

4.2.4. Set Show Name

A show is identified by a combination of creation time and username. If a name is given when the user creates the show, the show is listed by its name; otherwise, the software will generate a unique identifier for the show.

4.2.5. Pairing an Image File with a Personality File

In this case, there is no show-default personality. Note that the user can specify either show-default personality or personality pointer (See Sectionss 6 and 7 for how to define these files.)

When the data source is a directory or a stream, if there is a default personality, it will be used; otherwise, a personality will be chosen according to the personality pointer file.

The data files listed in a list can have an optional personality file assigned to it. The image file is listed with this personality file; however, if there is no personality file given, then the default personality file will be used.

4.2.6. List of shows

A user can create an arbitrary number of shows, private to the user. Created shows are listed in a table at the top of the page. Each row corresponds to a show that can be played and deleted. A show is identified by a combination of creation time and username. If a name is given when the user creates the show, the show is listed by its name; otherwise, the software will generate a unique identifier for the dhow.

In the current implementation, shows persist only in server instance memory and will disappear after a server reboot. It is planned to make them survivable over server reboots.

4.3. Browse Show (currently disabled)

To browse a show, click on the "browse" link. A "folder-like" page view of the show will be presented.

On this page, each member entry corresponds to a EDR file, which, in turn, can be clicked on and "opened" like a "folder". This process is traversing till to the end, when there is no more links. On each "folder-like" page, there are also links to meta-info in JSON format, which are helpful to developers who are creating new clients.

Technically, the "folder-like" page view of a show is the simplest html view of a webified "data store", which, in this case, is a show.

4.4. Play Show

To play a show, click on the "play" link. A new browser window will be spawned. In this new window, EDR files contained in the show are displayed one by one in default frequency set up when the show was created. To change the display speed, or tune the show in general, users can simply click anywhere in the window

to hide the "slide show". Now in the same window, the form for modifying show behavior will appear, along with show parameters. Try to change some, and click on “Watch The Show”, the change will take effect right way.

4.5. Delete Show

To delete a show, just click on the "delete" link from the list page.

4.6. Logout

A user logs out by clicking on the "logout" link. In current implementation, a logged-in user will be auto-logged out after a period of inactivity. The default idle time is 10 min (600 sec). When logged out, whether auto or not, created shows will still be alive.

5. Personality and Personality Pointer Files

Personality files are simply ASCII text files that contain component parameters that tell the server how the data will be displayed in the JEDI display window. The file can contain JEDI window component parameters or can be a list of pointers to other personality files to be used when displaying images. The client and server will reference and adhere to this file in presenting the EDR data to the user. The personality file must be viewable by the tomcat server. All parameters are optional except the PersonalityName.

5.1 Notations Used

The lines beginning with ‘#’ are comments.

5.2 Personality Types

There personality file can either be of type PERSONALITY or PERSONALITY-POINTER.

A PERSONALITY type file contains the parameters for formatting the display. The first line of this type of file should be exactly like this:

```
# Content-type: PERSONALITY
```

A PERSONALITY-POINTER file is a file containing references to other PERSONALITY files and controls the number of instances the personality is used. The first line of a personality pointer file should be exactly like this:

```
# Content-type: PERSONALITY-POINTER
```

Personalities are required to have a version number, which can be specified by beginning a line with “# version”. Other lines that start with a pound (#) sign is a comment, and will be ignored by the personality parser. On the other hand, personality pointers are not to have a version or comments.

6. Personality Parameters

The following sections list the parameters for a personality file. The mandatory property is PersonalityName. Other ones are optional.

Syntax for the property names and values are as follows:

- Property name is in camel case and case-sensitive.
- Property values take in standard css value if applicable. For possible values, please check any good online css reference such as http://www.w3schools.com/css/css_reference.asp

6.1 Personality Name

Each personality file must have a personality name usually the name of the instrument and will be used to match the role that the user has been assigned to. The name is case insensitive. The default name is PROJ.

```
PersonalityName=PROJ
```

6.2 Display Geometry

The DisplayGeometry allows the user to specify the location and dimensions of the JEDI window.

- **x** and **y** are attributes that tell the server to display the JEDI window at the **'x'** row, **'y'** column position on the User's Desktop Display.
- **width** and **height** are attributes that tell the server to display the JEDI window of the provided width and height sizes in pixels.

```
DisplayGeometry[x=10, y=100, width=620, height=600]
```

6.3 InstrumentDNScale

InstrumentDNScale specifies how to scale the pixel data number values (DNs) into a 0-256 range.

- The **short** attribute tells JEDI how many bits are used for the DN range from the instrument. This helps in scaling as many cameras of late use 12-bit ranges. Since such values are stored as 16-bit shorts, the short=12 specification helps JEDI knows how many bits are actually used to convey the DN from the instrument readings.
- The **byte** attribute is for the same use. Typically, this will always be set to 8. It is included here simply for completeness. Also, if ever there were an instrument that used less than a bytes worth of DN range, this might be helpful.
- You may enter a value of -1 for to **byte** and **short** if the user desires JEDI to perform auto scaling, i.e., let the software figure out the range and scaling.

```
InstrumentDNScale[byte=8, short=12]
```

6.4 IgnoreValues

IgnoreValues specify which pixel data number values (DNs) will be ignored when

calculating the mean, max, min values etc. This enhances the quality of the picture by ignoring the saturated points, for example. You may have up to 10 ignored values. Index starts from 1.

```
IgnoreValues[1=4095, 2=255]
```

6.5 Ignore Range

IgnoreRange is for specifying a bigger range of pixel data number values (DNs) to be ignored. Any value <= the lower value or >= the upper value will be ignored.

```
IgnoreRange[lower=0, upper=4095]
```

6.6 EDRCanvas

EDRCanvas specifies the information related to the EDR:

- The **name** attribute is the name of the EDR canvas
- The **x, y, width**, and **height** attributes specify the geometry of the canvas.
- The **stretchType** attribute can be percent or linear, and is the stretch method used.
- The **param0** and **param1** attributes are the lower and upper limits fo the stretch.
- The following 5 attributes are optional, but must be specified as a group. If you just specify 1 to 4 parameters, the special check is not performed.
    - **minBelowBg** specifies the absolute minimum.
    - **minLowerLimit**
    - **minDifference**
    - **minAbsolute**
    - **maxAboslute**
- The following 4 attributes are optional, but must be specified as a group. i.e., if you just specify 1 to 3 parameters, the special check is not performed.
    - **missingLineDN** specifies the value that indicates a missing line.
    - **missingLineR**, **missingLineG**, and **missingLineB** are the RGB values for the pixel with with a matching missingLineDN.

```
EDRCanvas[name=iss, x=30, y=10, width=256, height=256,
stretchTyp = percent, \param0=0.001, param1=0.001,
minBelowBg=15.0, minLowerLimit=30.0, minDifference=50.0,
\minAbsolute=0.0, maxAbsolute=4095.0, missingLineDN=0,
missingLineR=255, missingLineG=255, \missingLineB=0]
```

6.7 GreyWedge

GreyWedge is a grey bar that shows the gradiation of the DN color scale from 0 to 255.

- The **x, y, width**, and **height** attributes specify the geometry of the canvas.
- **ascending** steps of gray in the wedge. false means descending.

- **vertical** display of the grey wedge*.* false means a horizontal display.
- **steps** is the number of steps shows in the grey wedge, value is between 1 and 256

```
GreyWedge[name=iss, x=11, y=10, width=16, height=256,
ascending=false, vertical=true, steps=256]
```

6.8 HistogramCanvas

HistogramCanvas specifies the information related to the histogram:

- **name** is the name of the histogram
- **x, y, width** & **height** specifies the geometry of the canvas.
- **axis** specifies whether the histogram is shown as log or linear.

```
HistogramCanvas[name=iss, x=330, y=480, width=256,
height=100, axis=log]
```

6.9 IconCanvas

IconCanvas specifies the information related displaying an icon:

- **x & y** specifies the location of the icon
- **URL** specifies the location of the image used for the icon

```
IconCanvas[x=520,y=10,URL=[https://tpsweb.jpl.nasa.gov/Jedi/i
mages/casslogo_4.gif]
```

6.10 KeywordCanvas

KeywordCanvas specifies the information related to displaying a keyword:

- **x** and **y** specifies the location of the keyword
- **keyword** is the tag used to match the keyword in the EDR label for retrieving the value
- **value** is a place holder to indicate that there's a value associated w/ this keyword
- **show** is the text for the keyword that is displayed
- **delimiter** specifies the symbol used to separate the keyword and its value.
- **color** specifies the color of the text
- **alarmValue** & **alarmColor** specifies the special alarmed condition
- **alarmOp** is the alarm operation that will be used to determine if the current value is in alarm condition.
    - 0 means no comparison,1 is for =, 2 is for !=, 3 is for >, 4 is <.
- **evaluatedAsString** is set to false if the comparison is not on numbers; otherwise it's true.
- **fontsize** specifies the font size

```
KeywordCanvas[x=330,y=20,keyword=ANTIBLOOMING_STATE_FLAG,valu
e=---
,show=Antiblooming,\delimiter=:,color=FFFF00,alarmValue=,alar
mColor=FF0000,alarmOp=0,evaluatedAsString=true,\fontSize=14]K
eywordCanvas[x=330,y=40,keyword=CALIBRATION_LAMP_STATE_FLAG,v
alue=---,show=Cal.
Lamp,\delimiter=:,color=FFFF00,alarmValue=,alarmColor=FF0000,
alarmOp=0,evaluatedAsString=true,\fontSize=14]
```

6.11 Additional Property Values

The following table lists the additional values for the given properties:

Property Name	Additional Property Values
DisplayGeometry	backgroundColor, backgroundImage, backgroundRepeat
EDRCanvas	maskBorderThickness, maskBorderColor
IconCanvas	Property URL
KeywardCanvas	a keyword group leader must have property group
KeywordCanvas	a keyword group leader can have properties color, fontSize, fontWeight, fontFamily and align. Please note that align is not a css property and its value can be one of left, right, delimiter. left is the default

7. Personality Pointer Files

The personality pointer files references other personality files. Each line, after the first line, in the file can either be a blank line, or a pair of number and absolute file paths. The show will rotate through the list of given personality files to be applied to the image source.

Personality Pointer Example:

```
# Content-type: PERSONALITY-POINTER
3 /personalities/per_1.txt
2 /personalities/per_2.txt
```

In this example, the first three slides of the show will be displayed using per_1.txt, the next two slides will be displayed using per_2.txt, the next three with per_1.txt, the next two with per_2.txt, and so forth. This rule holds true whether the show is looping or not.

8. Application Features

8.1 Mini Show Meta Info

In the “My Shows” section of the list.html page, users can hover over the show’s “`play`” or “`delete`” links to see the show’s creator and its source displayed near the mouse pointer.

8.2 Looping

If the data source is a directory or a list file, the show can display the data again and again once it has displayed all of the data. This option is not available for stream source.

To activate the option, please choose “Loop through files” when creating the show.

8.3 Start Show with Latest Data

If the data source is a stream, the play of the show always starts with the latest data item in the stream and continues to only display data that comes later.

To activate this option, please choose “Start with current EDR” when creating the show.

8.4 Sample Rate

Sample rate is the time a slide is displayed until it’s replaced by the next slide. Users can change this by clicking on the slide to get to its control panel. The acceptable range is 1-50. However, please note that if you have a slow client and choose a sample rate of 1, the client may not be able to refresh quick enough and may result in missed images.

8.5 Stretch Parameter and Band

Stretch Parameter and Band are display parameters of an EDR. Changing these parameters affect the image display of that EDR. Each slide can contain more than one EDR. For example, Cassini’s ISS slide contains one EDR per slide while Cassini’s VIMS slide contains two EDRs per slide. Consequently, there should be as many Stretch Parameter controls as there are EDRs. In the case of VIMS, this rule also applies to Band controls.
To get to the control, users can click on the slide to get to the control panel.

8.6 Auto Window Resize

By the default, for each slide, the play window will automatically resize to the DisplayGeometry specified in the personality of the slide. To deactivate this feature, users can click on “Turn resize off” in the control panel. To reactivate the feature, users can click on “Turn resize on” in the control panel.

8.7 Debug

To inspect which data file is being displayed and how many slides have been displayed, users can click on “Turn debug on” in the control panel. Once debug is on, the data file’s name and the total number of slides so far is displayed on the top left of the play window.

8.8 Help

For help to use JEDI, users can click on the “Help” link at the top of list.html.

9. Input Update

Input to a show is both the data source and the personalities. This section

discusses how the client software handles changes in the show's input.

9.1 Directory Source Update

When data is added, deleted, or modified in the directory, the changes should be reflected in the show after a twenty-second delay.

9.2 List Source Update

Similarly to a directory source, when EDR-personality pairs are added or deleted in the list, the list is modified entirely, or items are no longer paired with the same personality, or any personality at all, the changes should be correctly reflected after a twenty-second delay.

9.3 Stream Source Update

Unlike directory or list source, stream source only has added data instead of deleted data. However, new data can be added to a stream at random time before the stream expires. As long as the show is running, new data should be displayed as soon as the display rate allows unless there is data that has not yet displayed.

9.4 Personality Update

JEDI currently does not detect personality updates.

9.5 User's Modifications to Show

Users' changes to Stretch Parameter and Band are not persistent to personality file or to the show's internal data. Whenever a show is played, it starts with the same values for Stretch Parameter and Band. Moreover, changes to Stretch Parameter and Band while the show is playing should be reflected as soon as the next slide that uses the same personality.

10. Known Issues

10.1 Stalking Play Window

When auto-resizing feature of the play window is turned on, if the user minimizes the play window, it will un-minimize as soon as the next slide.
Moreover, if users are using Common Desktop Environment (CDE), the play window will follow them to their current workspace. To resolve this, turn off the resizing after the show has been created.

10.2 Stale Play Window

When connection to the server is lost long enough while a show is playing, the play window gets stuck at its last slide and won't close nor move on. Connection loss may due to server down, client's Ethernet cable coming loose, or router being reset.

10.3 Persistent Debug Info

Occasionally, it takes two or three tries to turn off debug option in the play window.

10.4 Display Rate

If the display rate is set higher than 3, i.e. display 1 slide every 3 seconds, sometimes slides will be missed because the server is told to create a new slide while creating the current one.

Appendix A: Example of a Cassini Personality file

```
# Content-type: PERSONALITY
# version 1.1.0
# Each personality file must contain a PersonalityName.
# This is usually the name of the instrument.
# It will be used to match the role that the user has been assigned to.
PersonalityName=CASISS

# DisplayGeometry lets the user specifies the x, y location of the
# JEDI window (applicable for stand-alone version only) and the
# width and height of the window.
DisplayGeometry[x=10, y=100, width=620, height=600]

# InstrumentDNScale specifies how to scale the DN's into a 0-256 range.
# The short specification tells JEDI how many bits are used for the DN range
# from the instrument. This helps in scaling as many cameras of late use
# 12-bit ranges. Since such values are stored as 16-bit shorts, the short=12
# specification helps JEDI knows how many bits are actually used to convey the
# DN from the instrument readings. The byte specification is for the
# same use. Typically, this will always be set to 8. It is included here
# simply for completeness. Also, if ever there were an instrument that used
# less than a byt'es worth of DN range, this might be helpfu.
# You may enter a -1 for auto scaling, i.e., let the software figure out
# the range and scaling.
InstrumentDNScale[byte=8, short=12]

# IgnoreValues specify which values will be ignored when calculating
# the mean, max, min values etc. This enhanced the quality of the
# picture by ignoring the saturated points, for example. You may
# have up to 10 ignored values. Index starts from 1.
IgnoreValues[1=4095, 2=255]

# IgnoreRange is for specifying a bigger range of values to be
# ignored. Any value <= the lower value or >= the upper value will
# be ignored.
IgnoreRange[lower=0, upper=4095]

# EDRCanvas specifies the information related to the EDR:
#   name is the name of the EDR canvas
#   x, y, width, & height specifies the geometry of the canvas.
#   stretchType can be percent or linear, and is the stretch method used.
#   param0 and param1 are the lower and upper limits fo the stretch.
# The following 5 parameters are optional, but must be specified as a group.
i.e.,
# if you just specify 1 to 4 parameters, the special check is not performed.
#   minBelowBg specifies the absolute minimum.
#   minLowerLimit
#   minDifference
#   minAbsolute
#   maxAboslute
# The following 4 parameters are optional, but must be specified as a group.
i.e.,
# if you just specify 1 to 3 parameters, the special check is not performed.
#   missingLineDN specifies the value that indicates a missing line.
#   missingLineR, missingLineG, & missingLineB are the RGB values for the pixel
w/
#    with a matching missingLineDN.
EDRCanvas[name=iss, x=30, y=10, width=256, height=256, stretchTyp = percent, \
param0=0.001, param1=0.001, minBelowBg=15.0, minLowerLimit=30.0,
minDifference=50.0, \
minAbsolute=0.0, maxAbsolute=4095.0, missingLineDN=0, missingLineR=255,
```

```
missingLineG=255, \
missingLineB=0]

# GreyWedge is a grey bar that shows the graduation??
GreyWedge[name=iss, x=11, y=10, width=16, height=256, ascending=false,
vertical=true, steps=256]

# HistogramCanvas specifies the information related to the histogram:
#   name is the name of the histogram
#   x, y, width & height specifies the geometry of the canvas.
#   axis can be log or linear. It specifies whethe the histogram is shown as log
or linear.
HistogramCanvas[name=iss, x=330, y=480, width=256, height=100, axis=log]

# IconCanvas specifies the information related to the icon:
#   x & y specifies the location of the icon
#   URL specifies the location of the image used for the icon
IconCanvas[x=520, y=10,
URL=https://tpsweb.jpl.nasa.gov/Jedi/images/casslogo_4.gif]

# KeywordCanvas specifies the information related to a keyword:
#   x & y specifies the location of the keyword
#   keyword is the tag used to match the keyword in the EDR label for retrieving
the value
#   value is a place holder to indicate that there's a value associated w/ this
keyword
#   show is the text for the keyword that is displayed
#   delimiter specifies the symbol used to spearate the keyword and its value.
#   color specifies the color of the text
#   alarmValue & alarmColor specifies the special alarmed condition
#   alarmOp is the alarm operation that will be used to determine if the current
value
#     is in alarm condition. 0 means no comparison, 1 is for =, 2 is for !=, 3 is
for >,
#     4 is <.
#   evaluatedAsString is set to false if the comparison is not on numbers;
#     otherwise it's true.
#   fontsize specifies the font size
KeywordCanvas[x=330,y=20,keyword=ANTIBLOOMING_STATE_FLAG,value=---
,show=Antiblooming,\

delimiter=:,color=FFFF00,alarmValue=,alarmColor=FF0000,alarmOp=0,evaluatedAsStrin
g=true,\
   fontSize=14]
KeywordCanvas[x=330,y=40,keyword=CALIBRATION_LAMP_STATE_FLAG,value=---,show=Cal.
Lamp,\

delimiter=:,color=FFFF00,alarmValue=,alarmColor=FF0000,alarmOp=0,evaluatedAsStrin
g=true,\
   fontSize=14]
```

Appendix B: Example of a Personality Pointer file

```
# Content-type: PERSONALITY-POINTER
3 /full/path/to/personalities/per_1.txt
2 /full/path/to/personalities/per_2.txt
```

Appendix C: Installation Procedures

1. Create group configuration file :

   - Create groups in JPL directory service. E.g. group 'iss_dev' for Cassini's ISS product.
   - Create a default personality file for your group.
   - Group config file is a text file where each line is a tuple of three space-separated and case-sensitive elements with the first token is the group name, the second token is the name of personality associated with the group, and the third and last element is the full file name of the default personality file for the group. Blank lines are ignored. Here is an example of a line in the group config file:

```
iss_dev CASISS
/Users/honghanh/Documents/workspace/jedi/data/personalities/ISS_labwide_B_1_test.txt
```

2. Get a copy of ehcache.xml:

   - A copy of ehcache.xml is available in Appendix D.
   - This ehcache.xml is a config file, and you are welcome to write your own using the given one here as an example. For more information on how to write this file, please contact Zhangfan Xing (his contact info is available on JPL Space).
   - Save it and take note of the location and reference it in the web.xml file.

3. Unpack WAR file in some temporary directory:

```
% mkdir jujube
% cd jujube
% jar -xf /path/to/jujube.war
```

4. Modify web.xml if using JPL LDAP for Authentication and Authorization

Context-param Name	**Value**
jujube.authentication.service	*jpl.mipl.jujube.auth.LdapAuthentication*
jujube.authentication.ldap	*ldap://ldap.jpl.nasa.gov:636* (this is JPL LDAP server)
jujube.authorization.ldap	*ldap://ldap.jpl.nasa.gov:389/ou=personnel,dc=dir,dc=jpl,dc=nasa,dc=gov* (this JPL LDAP server)
jujube.authorization.config,	*/*path/to/config
jujube.auth	*true*
jujube.admins	The JPL username of whomever in charge of content policies of the JEDI server. The value can be a comman-delimited lists of usernames.
param treevotee.simplelogger.level	*none, info, debug, warn, error, fine*
jujube.simplecache.enable	yes
jujube.simplecache.ehcache.xml.path	/path/to/ehcache.xml
jujube.ehcache.xml.path	/path/to/ehcache.xml

jujube.stream.idle	set value to the number of hours that your streams are allowed to be idle before it's removed from the list of streams. If this value is not set, the default value is 24 hours. A stream is idle when there is no item added to it, and no show is using it.

5. Repack your WAR file

   Inside the jujube directory that we created earlier, please do (your original WAR file will be overwritten): jar -cf ../jujube.war *

6. Add WAR file to webapps location

```
% cp jujube.war to /usr/share/tomcat5/webapps/
```

7. Turn on HTTPS:

   To turn on encryption, you should add this section after all servlet-mapping sections in web.xml:

```
<security-constraint>
        <web-resource-collection>
            <web-resource-name>Automatic SSL Forwarding</web-resource-name>
            <url-pattern>/*</url-pattern>
            </web-resource-collection>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
            </user-data-constraint>
        </security-constraint>
```

   You should also enable HTTP with encryption protocol in Tomcat's server.xml. Your sysadmin should know how to do this. All of your data, including username, password, images and keywords will be encrypted if you do this.

8. Path to Jedi/Jujube:

   http://host:port/jujube/login.html

Appendix D: Sample ehcache.xml File

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
CacheManager Configuration
==========================
An ehcache.xml corresponds to a single CacheManager.

See instructions below or the ehcache schema (ehcache.xsd) on how to configure.

System property tokens can be specified in this file which are replaced when the
 configuration
is loaded. For example multicastGroupPort=${multicastGroupPort} can be replaced
with the
System property either from an environment variable or a system property specifi
ed with a
command line switch such as -DmulticastGroupPort=4446.

The attributes of <ehcache> are:
* name - an optional name for the CacheManager.  The name is optional and primar
ily used
for documentation or to distinguish Terracotta clustered cache state.  With Terr
acotta
clustered caches, a combination of CacheManager name and cache name uniquely ide
ntify a
particular cache store in the Terracotta clustered memory.
* updateCheck - an optional boolean flag specifying whether this CacheManager sh
ould check
for new versions of Ehcache over the Internet.  If not specified, updateCheck="t
rue".
* monitoring - an optional setting that determines whether the CacheManager shou
ld
automatically register the SampledCacheMBean with the system MBean server.
Currently, this monitoring is only useful when using Terracotta clustering and u
sing the
Terracotta Developer Console. With the "autodetect" value, the presence of Terra
cotta clustering
will be detected and monitoring, via the Developer Console, will be enabled. Oth
er allowed values
are "on" and "off". The default is "autodetect". This setting does not perform a
ny function when
used with JMX monitors.
* dynamicConfig - an optional setting that can be used to disable dynamic config
uration of caches
associated with this CacheManager.  By default this is set to true - i.e. dynami
c configuration
is enabled.  Dynamically configurable caches can have their TTI, TTL and maximum
 disk and
in-memory capacity changed at runtime through the cache's configuration object.
-->
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ehcache.xsd"
         updateCheck="true" monitoring="autodetect"
         dynamicConfig="true" >

    <!--
    DiskStore configuration
    =======================

    The diskStore element is optional. To turn off disk store path creation, com
ment out the diskStore
    element below.
```

```
    Configure it if you have overflowToDisk or diskPersistent enabled for any ca
che.

    If it is not configured, and a cache is created which requires a disk store,
 a warning will be
     issued and java.io.tmpdir will automatically be used.

    diskStore has only one attribute - "path". It is the path to the directory w
here
    .data and .index files will be created.

    If the path is one of the following Java System Property it is replaced by i
ts value in the
    running VM. For backward compatibility these should be specified without bei
ng enclosed in the ${token}
    replacement syntax.

    The following properties are translated:
    * user.home - User's home directory
    * user.dir - User's current working directory
    * java.io.tmpdir - Default temp file path
    * ehcache.disk.store.dir - A system property you would normally specify on t
he command line
      e.g. java -Dehcache.disk.store.dir=/u01/myapp/diskdir ...

    Subdirectories can be specified below the property e.g. java.io.tmpdir/one

    -->
    <diskStore path="java.io.tmpdir"/>


    <!--
    TransactionManagerLookup configuration
    =======================================
    This class is used by ehcache to lookup the JTA TransactionManager use in th
e application
    using an XA enabled ehcache. If no class is specified then DefaultTransactio
nManagerLookup
    will find the TransactionManager in the following order

     *GenericJNDI (i.e. jboss, where the property jndiName controls the name of
the TransactionManager object to look up)
     *Websphere
     *Bitronix
     *Atomikos

    You can provide you own lookup class that implements the net.sf.ehcache.tran
saction.manager.TransactionManagerLookup interface.
    -->

    <transactionManagerLookup class="net.sf.ehcache.transaction.manager.DefaultT
ransactionManagerLookup" properties="" propertySeparator=":"/>

    <!--
    CacheManagerEventListener
    =========================
    Specifies a CacheManagerEventListenerFactory which is notified when Caches a
re added
    or removed from the CacheManager.

    The attributes of CacheManagerEventListenerFactory are:
    * class - a fully qualified factory class name
    * properties - comma separated properties having meaning only to the factory
.
```

```
    Sets the fully qualified class name to be registered as the CacheManager eve
nt listener.

    The events include:
    * adding a Cache
    * removing a Cache

    Callbacks to listener methods are synchronous and unsynchronized. It is the
responsibility
    of the implementer to safely handle the potential performance and thread saf
ety issues
    depending on what their listener is doing.

    If no class is specified, no listener is created. There is no default.
    -->

    <cacheManagerEventListenerFactory class="" properties=""/>

    <!--
    TerracottaConfig
    ========================
    (Enable for Terracotta clustered operation)

    Note: You need to install and run one or more Terracotta servers to use Terr
acotta clustering.
    See http://www.terracotta.org/web/display/orgsite/Download.

    Specifies a TerracottaConfig which will be used to configure the Terracotta
    runtime for this CacheManager.

    Configuration can be specified in two main ways: by reference to a source of
    configuration or by use of an embedded Terracotta configuration file.

    To specify a reference to a source (or sources) of configuration, use the ur
l
    attribute.  The url attribute must contain a comma-separated list of:
    * path to Terracotta configuration file (usually named tc-config.xml)
    * URL to Terracotta configuration file
    * <server host>:<port> of running Terracotta Server instance

    Simplest example for pointing to a Terracotta server on this machine:
    <terracottaConfig url="localhost:9510"/>

    Example using a path to Terracotta configuration file:
    <terracottaConfig url="/app/config/tc-config.xml"/>

    Example using a URL to a Terracotta configuration file:
    <terracottaConfig url="http://internal/ehcache/app/tc-config.xml"/>

    Example using multiple Terracotta server instance URLs (for fault tolerance)
:
    <terracottaConfig url="host1:9510,host2:9510,host3:9510"/>

    To embed a Terracotta configuration file within the ehcache configuration, s
imply
    place a normal Terracotta XML config within the <terracottaConfig> element.

    Example:
    <terracottaConfig>
        <tc-config>
            <servers>
                <server host="server1" name="s1"/>
                <server host="server2" name="s2"/>
            </servers>
```

```
            <clients>
                <logs>app/logs-%i</logs>
            </clients>
        </tc-config>
    </terracottaConfig>

    For more information on the Terracotta configuration, see the Terracotta doc
umentation.
    -->
    <terracottaConfig url="localhost:9510"/>

    <!--
    Cache configuration
    ===================

    The following attributes are required.

    name:
    Sets the name of the cache. This is used to identify the cache. It must be u
nique.

    maxElementsInMemory:
    Sets the maximum number of objects that will be created in memory

    maxElementsOnDisk:
    Sets the maximum number of objects that will be maintained in the DiskStore
    The default value is zero, meaning unlimited.

    eternal:
    Sets whether elements are eternal. If eternal,  timeouts are ignored and the
    element is never expired.

    overflowToDisk:
    Sets whether elements can overflow to disk when the memory store
    has reached the maxInMemory limit.

    The following attributes and elements are optional.

    overflowToOffHeap:
    (boolean) This feature is available only in enterprise versions of Ehcache.
    When set to true, enables the cache to utilize "off-heap" memory
    storage to improve performance. Off-heap memory is not subject to Java
    GC cycles and has a size limit set by the Java property MaxDirectMemorySize.
    The default value is false.

    maxMemoryOffHeap:
    (string) This feature is available only in enterprise versions of Ehcache.
    Sets the amount of off-heap memory available to the cache.
    This attribute's values are given as <number>k|K|m|M|g|G|t|T for
    kilobytes (k|K), megabytes (m|M), gigabytes (g|G), or terrabytes
    (t|T). For example, maxMemoryOffHeap="2g" allots 2 gigabytes to
    off-heap memory. In effect only if overflowToOffHeap is true.


    timeToIdleSeconds:
    Sets the time to idle for an element before it expires.
    i.e. The maximum amount of time between accesses before an element expires
    Is only used if the element is not eternal.
    Optional attribute. A value of 0 means that an Element can idle for infinity
.
    The default value is 0.

    timeToLiveSeconds:
    Sets the time to live for an element before it expires.
    i.e. The maximum time between creation time and when an element expires.
```

```
    Is only used if the element is not eternal.
    Optional attribute. A value of 0 means that and Element can live for infinit
y.
    The default value is 0.

    diskPersistent:
    Whether the disk store persists between restarts of the Virtual Machine.
    The default value is false.

    diskExpiryThreadIntervalSeconds:
    The number of seconds between runs of the disk expiry thread. The default va
lue
    is 120 seconds.

    diskSpoolBufferSizeMB:
    This is the size to allocate the DiskStore for a spool buffer. Writes are ma
de
    to this area and then asynchronously written to disk. The default size is 30
MB.
    Each spool buffer is used only by its cache. If you get OutOfMemory errors c
onsider
    lowering this value. To improve DiskStore performance consider increasing it
. Trace level
    logging in the DiskStore will show if put back ups are occurring.

    clearOnFlush:
    whether the MemoryStore should be cleared when flush() is called on the cach
e.
    By default, this is true i.e. the MemoryStore is cleared.

    memoryStoreEvictionPolicy:
    Policy would be enforced upon reaching the maxElementsInMemory limit. Defaul
t
    policy is Least Recently Used (specified as LRU). Other policies available -
    First In First Out (specified as FIFO) and Less Frequently Used
    (specified as LFU)

    Cache elements can also contain sub elements which take the same format of a
 factory class
    and properties. Defined sub-elements are:

    * cacheEventListenerFactory - Enables registration of listeners for cache ev
ents, such as
      put, remove, update, and expire.

    * bootstrapCacheLoaderFactory - Specifies a BootstrapCacheLoader, which is c
alled by a
      cache on initialisation to prepopulate itself.

    * cacheExtensionFactory - Specifies a CacheExtension, a generic mechansim to
 tie a class
      which holds a reference to a cache to the cache lifecycle.

    * cacheExceptionHandlerFactory - Specifies a CacheExceptionHandler, which is
 called when
      cache exceptions occur.

    * cacheLoaderFactory - Specifies a CacheLoader, which can be used both async
hronously and
      synchronously to load objects into a cache. More than one cacheLoaderFacto
ry element
      can be added, in which case the loaders form a chain which are executed in
 order. If a
      loader returns null, the next in chain is called.
```

Cache Event Listeners

All cacheEventListenerFactory elements can take an optional property listenFor that describes
which events will be delivered in a clustered environment. The listenFor attribute has the
following allowed values:

* all - the default is to deliver all local and remote events
* local - deliver only events originating in the current node
* remote - deliver only events originating in other nodes

Example of setting up a logging listener for local cache events:

```
<cacheEventListenerFactory class="my.company.log.CacheLogger"
    listenFor="local" />
```

Cache Exception Handling
++++++++++++++++++++++++

By default, most cache operations will propagate a runtime CacheException on failure. An
interceptor, using a dynamic proxy, may be configured so that a CacheExceptionHandler can
be configured to intercept Exceptions. Errors are not intercepted.

It is configured as per the following example:

```
  <cacheExceptionHandlerFactory class="com.example.ExampleExceptionHandlerFactory"
                                properties="logLevel=FINE"/>
```

Caches with ExceptionHandling configured are not of type Cache, but are of type Ehcache only,
and are not available using CacheManager.getCache(), but using CacheManager.getEhcache().

Cache Loader
++++++++++++

A default CacheLoader may be set which loads objects into the cache through asynchronous and
synchronous methods on Cache. This is different to the bootstrap cache loader, which is used
only in distributed caching.

It is configured as per the following example:

```
    <cacheLoaderFactory class="com.example.ExampleCacheLoaderFactory"
                                properties="type=int,startCounter=10"/>
```

XA Cache
++++++++

To enable an ehcache as a participant in the JTA Transaction, just have the following attribute

transactionalMode="xa", otherwise the default is transactionalMode="off"

Cache Writer
++++++++++++

A CacheWriter maybe be set to write to an underlying resource. Only one Cach

```
eWriter can be
    been to a cache.

    It is configured as per the following example for write-through:

        <cacheWriter writeMode="write-through" notifyListenersOnException="true"
>
            <cacheWriterFactory class="net.sf.ehcache.writer.TestCacheWriterFact
ory"
                                properties="type=int,startCounter=10"/>
        </cacheWriter>

    And it is configured as per the following example for write-behind:

        <cacheWriter writeMode="write-behind" minWriteDelay="1" maxWriteDelay="5
"
                     rateLimitPerSecond="5" writeCoalescing="true" writeBatching
="true" writeBatchSize="1"
                     retryAttempts="2" retryAttemptDelaySeconds="1">
            <cacheWriterFactory class="net.sf.ehcache.writer.TestCacheWriterFact
ory"
                                properties="type=int,startCounter=10"/>
        </cacheWriter>

    The cacheWriter element has the following attributes:
    * writeMode: the write mode, write-through or write-behind

    These attributes only apply to write-through mode:
    * notifyListenersOnException: Sets whether to notify listeners when an excep
tion occurs on a writer operation.

    These attributes only apply to write-behind mode:
    * minWriteDelay: Set the minimum number of seconds to wait before writing be
hind. If set to a value greater than 0,
      it permits operations to build up in the queue. This is different from the
 maximum write delay in that by waiting
      a minimum amount of time, work is always being built up. If the minimum wr
ite delay is set to zero and the
      CacheWriter performs its work very quickly, the overhead of processing the
 write behind queue items becomes very
      noticeable in a cluster since all the operations might be done for individ
ual items instead of for a collection
      of them.
    * maxWriteDelay: Set the maximum number of seconds to wait before writing be
hind. If set to a value greater than 0,
      it permits operations to build up in the queue to enable effective coalesc
ing and batching optimisations.
    * writeBatching: Sets whether to batch write operations. If set to true, wri
teAll and deleteAll will be called on
      the CacheWriter rather than write and delete being called for each key. Re
sources such as databases can perform
      more efficiently if updates are batched, thus reducing load.
    * writeBatchSize: Sets the number of operations to include in each batch whe
n writeBatching is enabled. If there are
      less entries in the write-behind queue than the batch size, the queue leng
th size is used.
    * rateLimitPerSecond: Sets the maximum number of write operations to allow p
er second when writeBatching is enabled.
    * writeCoalescing: Sets whether to use write coalescing. If set to true and
multiple operations on the same key are
      present in the write-behind queue, only the latest write is done, as the o
thers are redundant.
    * retryAttempts: Sets the number of times the operation is retried in the Ca
cheWriter, this happens after the
      original operation.
```

```
    * retryAttemptDelaySeconds: Sets the number of seconds to wait before retryi
ng an failed operation.

    Cache Extension
    +++++++++++++++

    CacheExtensions are a general purpose mechanism to allow generic extensions
to a Cache.
    CacheExtensions are tied into the Cache lifecycle.

    CacheExtensions are created using the CacheExtensionFactory which has a
    <code>createCacheCacheExtension()</code> method which takes as a parameter a
    Cache and properties. It can thus call back into any public method on Cache,
 including, of
    course, the load methods.

    Extensions are added as per the following example:

         <cacheExtensionFactory class="com.example.FileWatchingCacheRefresherExt
ensionFactory"
                               properties="refreshIntervalMillis=18000, loaderTime
out=3000,
                                           flushPeriod=whatever, someOtherProperty
=someValue ..."/>

    Terracotta Clustering
    +++++++++++++++++++++

    Cache elements can also contain information about whether the cache can be c
lustered with Terracotta.
    The <terracotta> sub-element has the following attributes:

    * clustered=true|false - indicates whether this cache should be clustered wi
th Terracotta. By
      default, if the <terracotta> element is included, clustered=true.
    * valueMode=serialization|identity - indicates whether this cache should be
clustered with
      serialized copies of the values or using Terracotta identity mode.  By def
ault, values will
      be cached in serialization mode which is similar to other replicated Ehcac
he modes.  The identity
      mode is only available in certain Terracotta deployment scenarios and will
 maintain actual object
      identity of the keys and values across the cluster.  In this case, all use
rs of a value retrieved from
      the cache are using the same clustered value and must provide appropriate
locking for any changes
      made to the value (or objects referred to by the value).
    * synchronousWrites=true|false - When set to true, clustered caches use
      Terracotta SYNCHRONOUS WRITE locks. Asynchronous writes (synchronousWrites
="false") maximize performance by
      allowing clients to proceed without waiting for a "transaction received" a
cknowledgement from the server.
      Synchronous writes (synchronousWrites="true") maximize data safety by requ
iring that a client receive server
      acknowledgement of a transaction before that client can proceed. If cohere
nce mode is disabled using
      configuration (coherent="false") or through the coherence API, only asynch
ronous writes can occur
      (synchronousWrites="true" is ignored). By default this value is false (i.e
. clustered caches use normal
      Terracotta WRITE locks).
    * coherent=true|false - indicates whether this cache should have coherent re
ads and writes with guaranteed
      consistency across the cluster.  By default, its value is true.  If this a
```

```
ttribute is set to false
      (or "incoherent" mode), values from the cache are read without locking, po
ssibly yielding stale data.
      Writes to a cache in incoherent mode are batched and applied without acqui
ring cluster-wide locks,
      possibly creating inconsistent values across cluster. Incoherent mode is a
 performance optimization
      with weaker concurrency guarantees and should generally be used for bulk-l
oading caches, for loading
      a read-only cache, or where the application that can tolerate reading stal
e data. This setting overrides
      coherentReads, which is deprecated.
    * copyOnRead=true|false - indicates whether cache values are deserialized on
 every read or if the
      materialized cache value can be re-used between get() calls. This setting
is useful if a cache
      is being shared by callers with disparate classloaders or to prevent local
 drift if keys/values
      are mutated locally w/o putting back to the cache. NOTE: This setting is o
nly relevant for caches
      with valueMode=serialization

    Simplest example to indicate clustering:
        <terracotta/>

    To indicate the cache should not be clustered (or remove the <terracotta> el
ement altogether):
        <terracotta clustered="false"/>

    To indicate the cache should be clustered using identity mode:
        <terracotta clustered="true" valueMode="identity"/>

    To indicate the cache should be clustered using incoherent mode for bulk loa
d:
        <terracotta clustered="true" coherent="false"/>

    To indicate the cache should be clustered using synchronous-write locking le
vel:
        <terracotta clustered="true" synchronousWrites="true"/>
    -->

    <!--
    Mandatory Default Cache configuration. These settings will be applied to cac
hes
    created programmtically using CacheManager.add(String cacheName).

    The defaultCache has an implicit name "default" which is a reserved cache na
me.
    -->
    <defaultCache
           maxElementsInMemory="0"
           eternal="false"
           overflowToDisk="true"
           timeToIdleSeconds="1200"
           timeToLiveSeconds="1200">
      <!-- <terracotta/> -->
    </defaultCache>

    <!--
    Sample caches. Following are some example caches. Remove these before use.
    -->

    <!--
    Sample cache named sampleCache1
    This cache contains a maximum in memory of 10000 elements, and will expire
```

```
    an element if it is idle for more than 5 minutes and lives for more than
    10 minutes.

    If there are more than 10000 elements it will overflow to the
    disk cache, which in this configuration will go to wherever java.io.tmp is
    defined on your system. On a standard Linux system this will be /tmp"
    -->
    <cache name="sampleCache1"
           maxElementsInMemory="10000"
           maxElementsOnDisk="1000"
           eternal="true"
           overflowToDisk="true"
           diskPersistent="true"
           diskSpoolBufferSizeMB="20"
           memoryStoreEvictionPolicy="LFU"
            />
<!--
           timeToIdleSeconds="300"
           timeToLiveSeconds="600"
-->


    <!--
    Sample cache named sampleCache2
    This cache has a maximum of 1000 elements in memory. There is no overflow to
 disk, so 1000
    is also the maximum cache size. Note that when a cache is eternal, timeToLiv
e and
    timeToIdle are not used and do not need to be specified.
    -->
    <cache name="sampleCache2"
           maxElementsInMemory="10000"
           maxElementsOnDisk="10000"
           eternal="true"
           overflowToDisk="true"
           diskPersistent="true"
           diskSpoolBufferSizeMB="20"
           memoryStoreEvictionPolicy="FIFO"
            />


    <!--
    Sample cache named sampleCache3. This cache overflows to disk. The disk stor
e is
    persistent between cache and VM restarts. The disk expiry thread interval is
 set to 10
    minutes, overriding the default of 2 minutes.
    -->
    <cache name="sampleCache3"
           maxElementsInMemory="500"
           eternal="false"
           overflowToDisk="true"
           timeToIdleSeconds="300"
           timeToLiveSeconds="600"
           diskPersistent="true"
           diskExpiryThreadIntervalSeconds="1"
           memoryStoreEvictionPolicy="LFU"
            />

    <cache name="oneSimpleCache"
           maxElementsInMemory="10000"
           maxElementsOnDisk="1000"
           eternal="true"
           overflowToDisk="true"
           diskPersistent="true"
```

```
            diskSpoolBufferSizeMB="20"
            memoryStoreEvictionPolicy="LFU"
             />
</ehcache>
```